基于逻辑一致性判定的广义不透明谓词检测方法 *

史大伟，周季璇，徐良华

(江南计算技术研究所，江苏 无锡 214083)

摘 要：不透明谓词是一类轻量级的代码混淆方法，能以单向的执行复杂度对抗程序的逆向分析。广义不透明谓词扩展狭义不透明谓词的值恒定属性至逻辑恒定属性，已经应用于部分恶意代码中以提升抗查杀能力。为消除不透明谓词对程序恶意性判定的影响，以广义不透明谓词后趋依赖的属性为依据，结合逻辑恒定判定，提出了基于逻辑一致性的广义不透明谓词检测方法。通过静态分析提取谓词前置条件约束、后趋逻辑约束和谓词判定表达式，以相交基本块搜寻初筛谓词，并依据约束求解方法判定广义不透明谓词。构造原型系统并进行测试，结果表明该方法能精准高效地检测出恶意代码中的不透明谓词。

关键词：不透明谓词；约束求解；执行逻辑；后趋约束

中图分类号：TP309.2　　doi: 10.3969/j.issn.1001-3695.2017.12.0824

Generalized opaque predicates detecting method based on logical consistency

Shi Dawei, Zhou Jixuan, Xu Lianghua

(*Jiangnan Institute of Computing Technology*, *Wuxi Jiangsu* 214083, *China*)

Abstract: Opaque predicate is a lightweight obfuscation method which holds partial observability and is used to impede reverse engineering. Generalized Opaque Predicate extends the property of narrow Opaque Predicate by turning fixed value to fixed logic, and it is applied in malware. In order to eliminate the disturbance introduced by opaque predicates during malware identifying, a generalized opaque predicate detecting method is proposed based on the consistency of logic, this method depends on the reliance on constraint, and combines with the identification of consistency toward logic. Our method extracts previous constraint of domain, back constraint of logic and expression of predicate, then filters candidates by applying search of intersecting basic blocks, and finally identifies opaque predicates through constraint solving. We designed a prototype and the evaluation indicates that our method could identify opaque predicates from malware accurately and effectively.

Key words: opaque predicate; constraint solving; execution logic; post-constraint

0 引言

不透明谓词是一类取值恒定的约束表达式，其取值在作用域条件下将始终保持不变，这种特性保证了不透明谓词较强的反分析能力，所以被广泛应用在各种抗逆向工程[1]的实践中。在约束表达式的生成中，不透明谓词使用复杂的数学算法进行精巧的设计，以获取单向复杂的表达式，常用的方法包括有代数定理、数学剩余理论等。这种单向复杂的逻辑特点，保证了不透明谓词的判断值对于使用者而言是可知的，而对于分析对象，则是无法简单判定的[2]，这也是不透明谓词的实践应用原理。

代码混淆技术用于保护程序关键位置的代码，通过对数据和逻辑的变换，来掩盖真实的程序执行流程，以抵御对程序的逆向分析。代码混淆包括了对源代码的混淆和对二进制代码的混淆，应用的混淆方法包括动态混淆方法和静态混淆方法。常用的混淆技术有不透明谓词、控制流平展[3]、基本块分裂、别名混淆等。其中，基于不透明谓词的代码混淆技术具有较小的体量和封闭的逻辑，相较于其他的混淆方法，更为轻量和稳定。从 1997 年，Collberg[4]首次提出不透明谓词的概念以来，不透明谓词已经被广泛应用在程序开发应用的多种领域，包括软件多样化、恶意代码变形、软件水印[5]等。尤其是在恶意代码混淆领域，不透明谓词通过简单的插入判定式，引入冗余的或者不可达的分支路径，复杂化程序控制流图，达到隐藏恶意行为的目的。

随着在恶意代码中的不断多样化和复杂化应用，对不透明谓词的检测获得了越来越多的关注。Collberg 首先提出了不透明谓词的检测方法[4]，将在大量测试例运行下保持恒定的谓词表达式判定为不透明谓词，该方法采用的全程序遍历的思想耗

收稿日期：2017-12-26；**修回日期：**2018-02-27　**基金项目：**国家“863”计划资助项目（2012AA7111043）；国家自然科学基金资助项目（91318301）

作者简介：史大伟（1989-），男，江苏金坛人，博士研究生，主要研究方向为信息安全（sdave@126.com）；周季璇（1993-），女，硕士研究生，主要研究方向为软件测试；徐良华（1963-），男，高级工程师，博士，主要研究方向为信息安全.

时较大。Preda 通过抽象解释方法搜寻不透明谓词的代码特征[6]，而后对符合条件的位置进行鉴别，该方法只能适用于已知模式的不透明谓词检测。Madou 通过 Fuzzing 的方法对谓词的恒值性进行测定，也存在时间消耗大的缺点。Saleh 提出了通过检查在反汇编的代码中存在的交错状的指令结构来检测不透明谓词的方法[7]，但该方法抗干扰能力较差，会产生很多的误报。相对而言，Bardin 提出的有界前溯方法在不透明谓词的检测中更具有通用性[8]，该方法使用动态符号执行的方法，以约束求解判定谓词表达式的恒值特征。JiangMing 同样提出了基于符号执行的不透明谓词判定方法，并提出了动态不透明谓词的概念[9]，但该谓词实际上是狭义不透明谓词的特殊表示。

由上可见，当前不透明谓词判别方法主要集中于对单个逻辑判断式取值恒定的检测，对于复杂的逻辑结构群，并未设计有效的检测方法。鉴于此，本文分析了不透明谓词的主要机理，提炼和狭义不透明谓词和广义不透明谓词两种形式，并重点针对广义不透明谓词，提出了对应的检测方法。主要的技术原理是结合约束求解技术，确定并分析表达式的逻辑一致性，该方法将有效的拓展不透明谓词的检测领域，更好的简化代码控制流图，更为高效的还原出恶意代码的执行逻辑。

1　狭义不透明谓词和广义不透明谓词

作为一种精心构造的约束表达式，不透明谓词 P 的表达式值输出 O(P)一般为固定常量值，但是在某些条件下，不透明谓词的输出为逻辑表达式。以图 1 为例，在左侧的判断约束表达式中，$7y^2-1\neq x^2$ 的判定取值始终为 True，在此条件下跳转执行的路径将恒定为 Ins1。而在右侧的判断约束表达式中，$x+y==0$ 的判定取值可为 True，亦可为 False，当跳转至 True 路径时，执行的逻辑为 $z=2x$，当跳转至 False 路径时，执行的逻辑为 $z=-2y$，在此逻辑结构的条件下，进行值域取值分析，在 True 路径，条件约束为 $x+y==0$ 时，等式 $x=-y$ 成立，则有表达式 $z=2x=-2y$，故在 True 路径，执行逻辑依旧为 $z=-2y$，可得在当前逻辑约束群结构中，路径的跳转并未改变代码的执行逻辑。本文将这类在执行逻辑无区别的不透明谓词定义为广义的不透明谓词，而将传统的不透明谓词定义为狭义不透明谓词，二者在差异在于约束表达式的输出不同。

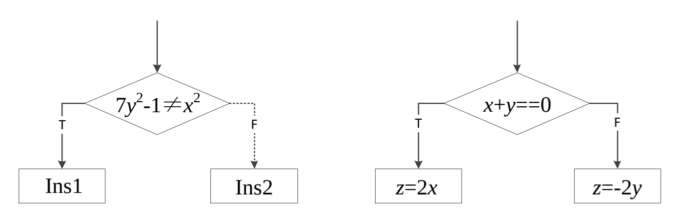


图 1　狭义不透明谓词和广义不透明谓词的跳转图示

a)狭义不透明谓词。狭义不透明谓词是一类侧重于取值不透明性的谓词表达式。狭义不透明谓词的表达式取值将始终恒定为固定值，该取值的固定特性用以保证程序能按照开发者指定的路径执行，而不会去执行另一条误导路径。狭义不透明谓词的不透明性是由跳转位置的谓词逻辑表达式和表达式参数的约束取值区间决定的。在此以约束表达式 $x^2-9>0$ 为例，其判定值为非固定，但是如若在之前的执行中，已对 x 的取值限制了定义域，如 $x>3$，则该谓词表达式的取值为将为恒定值，即可认定其不透明性。

b)广义不透明谓词。广义不透明谓词是一类侧重于逻辑不透明性的谓词表达式。广义不透明谓词的表达式取值并非为恒定值，但是结合该谓词表达式控制判定跳转后续的执行逻辑，整个逻辑结构群的行为逻辑具有恒定的特性。广义不透明谓词的不透明性是由跳转位置的逻辑表达式、表达式参数以及跳转路径上的逻辑表达式决定的。

对比发现，狭义不透明谓词的恒值特性来自于表达式本身和前置的表达式定义域约束；而广义不透明谓词的恒逻辑特性不单来自于表达式本身和前置的表达式定义域约束，并强相关于后趋的跳转执行逻辑。所以对广义不透明谓词的判定，相较于狭义不透明谓词的判定，需要扩展对后趋约束的分析。

表 1　典型的广义不透明谓词

谓词表达式	前置条件	路径分支	后趋路径逻辑	约束区间
$x^2-2x<2$	none	True	$y=x^3-2x^2$	$0\leq x\leq 2$
		False	$y=x^2-2x$	$x>2\vee x<0$
$x+y\leq 3$	$0\leq x\leq y$	True	$xy+y^2$	$(x,y)=\{(0,3);(0,2);(1,2);(0,1)\}$
		False	$2x+y^2$	$y\geq 3\wedge 0\leq x\leq y$
$x==k^2$	$-5\leq k\leq 5$	True	$y=x/7$	$x=\{0,1,4,9,16,25\}$
		False	$y=x/8$	$x\neq\{0,1,4,9,16,25\}$
$x\%3==2$	none	True	$y=(x+2)\%3$	$x=3k+2$
		False	$y=(x^2+2)\%3$	$x=3k+1\vee x=3k$

2　广义不透明谓词的判定方法

针对狭义不透明谓词的判定方法已经有很多种，一般常采用的是约束求解[8][9]、抽象解释[6]的基本方法，其中约束求解是通过判定取值的固定特征来检测不透明性，抽象解释是通过判断不透明谓词的构造数学依据来确定不透明性。面向广义不透明谓词的判定同样可以使用以上两种技术，但是抽象解释技术需要有较为丰富的先验知识积累，一般用于检测已知类型的不透明谓词，所以在本文中，为提升通用性，将采用的是约束求解技术。

设定待分析的程序 P 中某位置存在谓词逻辑表达式 ψ，ψ 的取值为 $\mathcal{M}(\psi)$，当 $\mathcal{M}(\psi)$取值为 True 时，执行路径 $\mathcal{B}^T$，后趋的执行逻辑为 $\mathcal{L}^T$，当 $\mathcal{M}(\psi)$取值为 False 时，将执行路径 $\mathcal{B}^F$，后趋的执行逻辑为 $\mathcal{L}^F$。

谓词逻辑表达式 ψ，为变量的函数表达式，可以表示为 $\psi=f(\alpha,\beta,\gamma\ldots)$，其中 $\alpha,\beta,\gamma\ldots$为 ψ 中的变量，根据前置约束，变量 $\alpha,\beta,\gamma\ldots$均有定义域，以 D 表示变量定义域的集合，$D=\{D(\alpha), D(\beta), D(\gamma)\cdots\cdots\}$。如在图 1 例中 ψ 为 $f(x,y)$，即为 $x+y==0$，

x，y 为 ψ 中的变量，定义域集合 $\mathcal{D}$ 表示为$\{\mathbb{Z},\mathbb{Z}\}$，$\mathbb{Z}$ 为全体整数集合。

在当前定义下，判定一个不透明谓词为狭义不透明谓词的判定方法为

$$\forall\alpha,\beta,\gamma\ldots\in\mathcal{D}\ \ M(\psi)\equiv C,\quad C=\text{True or False} \tag{1}$$

而为广义不透明谓词的判定方法为

$$\forall\alpha,\beta,\gamma\ldots\in\{D(\alpha),D(\beta),D(\gamma)\ldots\}\ \ \mathcal{L}_T\equiv\mathcal{L}_F \tag{2}$$

在该判定方法中，可见任取变量值（该变量不局限于 ψ 中涉及的变量），在判定并完成路径跳转后，后续的执行逻辑都相同，这反映出路径跳转判定的约束表达式 ψ 的取值对程序的执行路径有影响，但是对程序的执行逻辑无影响，这也反映了不透明谓词的冗余性，当然这也是混淆方法需要达到的效果。

由式（2）进一步推导，将表达式转换与 ψ 相关联的判定方法，首先将 ψ 的判定值取值区间表示出来，由于 ψ 是关于变量 $\alpha,\beta,\gamma\ldots$的函数表达式，则计算 ψ 取值的定义域约束。若只考虑公式 ψ，当 ψ=true 时，获取 ψ 取值为 True 的定义域约束为 d^T，同样，计算出 ψ 取值为 False 的定义域约束为 d^F。此时，可获取在不透明位置，跳转至路径 $\mathcal{B}^T$ 和 $\mathcal{B}^F$ 的定义域约束为基于谓词逻辑表达式 ψ 的约束和前置定义域约束的交集：

$$\begin{cases}\alpha,\beta,\gamma\ldots\in\mathcal{D}\cap d^T & \text{jump to }\mathcal{B}_T\\ \alpha,\beta,\gamma\ldots\in\mathcal{D}\cap d^F & \text{jump to }\mathcal{B}_F\end{cases} \tag{3}$$

要保证在两条跳转路径上的执行逻辑一致，则不透明谓词必须满足条件：

$$\begin{cases}\text{if }\alpha,\beta,\gamma\ldots\in\mathcal{D}\cap d^T & \mathcal{L}_T\equiv\mathcal{L}_F\\ \text{if }\alpha,\beta,\gamma\ldots\in\mathcal{D}\cap d^F\end{cases}\ \text{or}\ \begin{cases}\text{if }\alpha,\beta,\gamma\ldots\in\mathcal{D}\cap d^T\\ \text{if }\alpha,\beta,\gamma\ldots\in\mathcal{D}\cap d^F & \mathcal{L}_F\equiv\mathcal{L}_T\end{cases} \tag{4}$$

该条件的含义是，当在跳转目的路径为 $\mathcal{B}^T$ 时，在当前约束条件下，程序执行逻辑 $\mathcal{L}^T$ 等同于执行路径 $\mathcal{L}^F$；或者当跳转目的路径为 $\mathcal{B}^F$ 时，在当前约束条件下，程序执行逻辑 $\mathcal{L}^F$ 等同于执行路径 $\mathcal{L}^T$，此时可判定当谓词为不透明。举例说明，在图 1 例中，当 ψ 中的变量 x,y 满足约束 $x+y==0$ 时，将执行路径 $\mathcal{B}^T$，这条路径条件下的执行逻辑为 $\mathcal{L}^T$，即为 $z=2x$，对比执行逻辑 $\mathcal{L}^F$：$z=-2y$，可以发现在当前条件约束$_{\mathcal{D}\cap d^T}$：$\{x\in\mathbb{Z},y\in\mathbb{Z},x=-y\}$下，执行逻辑 $\mathcal{L}^T$ 与 $\mathcal{L}^F$ 具有一致性，故可判定当前选定的谓词为广义不透明谓词。

在上述实例的分析中，可计算发现，虽然满足不透明谓词判定的第一部分条件，但是当执行路径 $\mathcal{B}^F$ 时，执行逻辑 $\mathcal{L}^T$：$z=-2y$ 在条件约束$_{\mathcal{D}\cap d^F}$：$\{x\in\mathbb{Z},y\in\mathbb{Z},x\neq-y\}$下，与执行逻辑 $\mathcal{L}^F$ 并不具有一致性。所以在对广义不透明谓词的判定中，只需遵循二者条件其一即可。

3 广义不透明谓词的检测过程

根据广义不透明谓词的判定方法，在判定中的三个要素为前置条件、后趋逻辑、谓词表达式。本章将从广义不透明谓词的判定三要素内容开展提取和分析。

3.1 前置条件约束的提取

谓词的前置条件约束限定了谓词表达式判定当中的变量定义域。不充分的谓词变量定义域的信息获取，会导致谓词判定中定义域过大，从而使得不透明谓词判定条件的不满足，从而出现漏报的情况。例如表 1 中的第二条，如果未能准确的判别出存在 $0\le x\le y$ 的前置条件约束，则 $xy+y^2$ 和 $2x+y^2$ 在当前定义域取值区间中，$x+y\le3$ 谓词表达式的判定为非不透明。

对前置条件约束的提取需要分析整个谓词位置之前执行的所有指令，从而去全面的判定定义域的区间。然而前置所有指令的分析，消耗的时间过大，从而极大影响不透明谓词的判定效率。本文将采用 K 基本块的前置条件约束判定方法[8]，这是根据程序执行逻辑和编译器优化的特点选定的方法，即在程序代码中，关联性强的变量总是在连续的块区间内进行操作。根据文章的实验，选定的回溯基本块长度 K 为 16 至 24 间为最佳，在本文的不透明谓词检测中，将以此为优选参数。

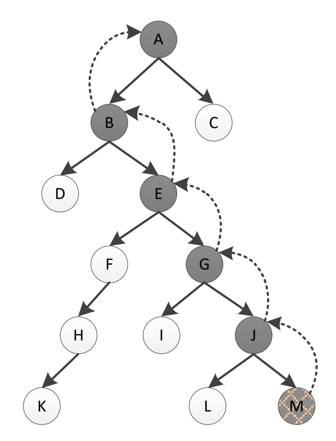


图 2 有界 K 基本块前置条件约束前溯示意图

在谓词表达式前置条件约束的提取过程为：

a）从选定的谓词出发，前向搜寻当前基本块的前置基本块节点，并形成长度为 K 的前置节点链。在图 2 中，每个圆形节点表示一个基本块，其树型结构是执行的控制流图，其中 M 节点的前置节点链为 M←J←G←E←B←A……。

b）从获取的前置节点链中，提取关联谓词表达式 ψ 中变量 $\alpha,\beta,\gamma\cdots$的操作指令，并提取指令的语义。

c）对变量的前置节点链中的指令语义进行公式化表示，并化简，获取变量的前置条件约束。

3.2 后趋逻辑约束的提取

谓词的后趋逻辑约束包括了该判定后的真值路径逻辑 $\mathcal{L}^T$ 和假值路径逻辑 $\mathcal{L}^F$。与前置条件约束类似，对于后趋逻辑约束的提取也存在提取区间过长的问题，要想完整的提取出两条路径的后趋约束，必须将从谓词位置至程序结束位置的所有执行代码逻辑均列入分析。本文基于不透明谓词的插入方法，结合恶意代码中，对不透明谓词插入的用途分析，采用了后向基本块有界条件下后趋逻辑约束提取方法。方法的提出基于以下两点考虑：

a）在恶意代码中，广义不透明谓词的插入用以生成干扰路

径，但是从代码量的角度，路径 $\mathcal{L}^{T}$ 和路径 $\mathcal{L}^{F}$ 一定会有交汇，以达到路径干扰的功能，而不是全路径冗余。

b）对于广义不透明谓词的后趋执行路径，为保证执行逻辑的一致性，在后趋执行过程中，较少有分支节点，而是采用直接跳转的方式，如果存在条件跳转，则极大可能为狭义不透明谓词。

一个典型的后趋执行代码的控制流图结构表示如图 3 所示。

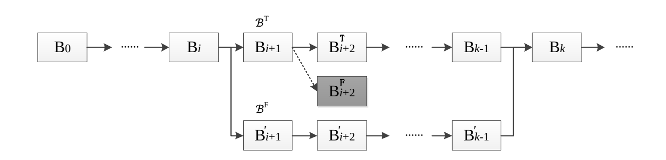


图 3　典型后趋执行代码控制流图结构

在图中，基本块以 B_i 表示，执行分支路径分别为 $\mathcal{B}^{T}$ 和 $\mathcal{B}^{F}$。其中在 B_{i+1} 中分支路径 B_{i+2} 为狭义不透明谓词。两条分支路径 $\mathcal{B}^{T}$ 和 $\mathcal{B}^{F}$ 在执行少量指令后相交并汇成同一条执行路径。本文提取的即是在 B_i 和 B_k 间的在 $\mathcal{B}^{T}$ 和 $\mathcal{B}^{F}$ 路径上的执行逻辑。在后趋逻辑约束的提取中，同样设定为 L 基本块长度的有界基本块分析方法，步骤如下：

a）从谓词跳转位置开始，搜寻分支路径 $\mathcal{B}^{T}$ 和 $\mathcal{B}^{F}$，在 L 基本块长度内寻找两条路径的相交节点，若未找到相交节点，则判定非不透明，若找到，进入下一步骤。

b）在谓词所在基本块 B_i 和相交节点基本块 B_k 间分析分析两条路径上的条件跳转节点，判定跳转节点的狭义不透明特性，若判定存在非不透明节点，则判定当前谓词逻辑为非广义不透明的，若无非不透明节点，进入下一步。

c）满足以上条件的路径分支，分别从 $\mathcal{B}^{T}$ 和 $\mathcal{B}^{F}$ 中提取基本块节点 B_i 和 B_k 间逻辑表达式。

3.3　基于约束求解的不透明性判定

对于不透明谓词的判定，是判断谓词的恒值性，对于广义不透明谓词，需要判断的是执行逻辑的恒定性。根据第三章的理论分析，主要的判定思想是证明在当前值域条件下，分支路径 $\mathcal{B}^{T}$ 和分支路径 $\mathcal{B}^{F}$ 上的逻辑可以统一表示为某一条路径上的执行逻辑。例如在第三章的实例分析中，该不透明谓词的后趋逻辑可以统一表示为 z=-2y，不管选择哪条分支路径，执行该逻辑表达式，均能获得一致的取值。

约束求解理论用以判定路径可达性并生成测试用例，主要依赖于可满足性模理论 SMT[10]。作为符号执行的基础，约束求解的性能直接影响了符号执行的效率。约束求解的核心问题是将路径条件中的算术约束条件转换为基本的求解器问题。本文采用的基于约束求解的不透明谓词判定方法即是将式(2)~(4)中的条件约束转换为可用于求解的问题，具体的转换方法是：找出路径 $\mathcal{B}^{T}$ 和路径 $\mathcal{B}^{F}$ 中相异的逻辑表达式 $\mathcal{F}^{T}$ 和 $\mathcal{F}^{F}$，判定以下表达式是否成立:

$$\forall\alpha,\beta,\gamma\ldots\in \mathcal{D}\cap d^{T}\ \ \mathcal{F}_{T}\equiv\mathcal{F}_{F} \parallel \forall\alpha,\beta,\gamma\ldots\in \mathcal{D}\cap d^{F}\ \ \mathcal{F}_{T}\equiv\mathcal{F}_{F}$$

判定值为真，则表示具有不透明性，为假，则表示不具备透明性。进一步转换，（$p\|q \Rightarrow \neg p\&\&\neg q$）将判定恒成立转变为判定是否存在不满足的解集，故判定表达式重写为：

$$(\exists\alpha,\beta,\gamma\ldots\in \mathcal{D}\cap d^{T}\ \ \mathcal{F}_{T}\neq\mathcal{F}_{F})\&\&(\exists\alpha,\beta,\gamma\ldots\in \mathcal{D}\cap d^{F}\ \ \mathcal{F}_{T}\neq\mathcal{F}_{F})$$

判定值为真，表示不具有透明性，而判定值为假，则具有透明性。同样以图 1 中的谓词逻辑作为判定实例。判定的表达式为：

$$(\exists x+y==0,\ 2x\neq-2y)\&\&(\exists x+y\neq0,\ 2x\neq-2y)$$

上述公式的判定值为假，可以确定该谓词逻辑结构群具有不透明性。

3.4　广义不透明谓词的判定算法

综上所述，本文提出基于约束求解的广义不透明谓词判定方法：

Input:　CFG: 程序 P 的控制流图

Output:　**B_i/F**: 若存在广义不透明谓词，则返回该谓词所在的基本块，若不存在，返回 False

```
Bi = breadthSearch (CFG)      //广度优先搜索控制流图
if  exist conditional jump in Bi      //提取条件跳转
  for p=1:L
    for q=1:L
      if  (followingBlock(Bi, p,True)==followingBlock(Bi, q,False))
//判定是否存在相交基本块
        ℒ ← extractLogic(Bi, p, True);
//提取跳转 True 路径上 Bi 至相交基本块间的逻辑表达式
        ℒ' ← extractLogic(Bi, p, False);
//提取跳转 False 路径上 Bi 至相交基本块间的逻辑表达式
        Ɗ ← extractDomain(Bi, K);
//提取前置定义域区间
        ψ ← extractFormula(Bi);
//提取条件跳转对应的谓词表达式
        if( constraintSolving(ℒ, ℒ', Ɗ, ψ));
//依照判定式约束求解，并返回结果
          return Bi;
        else  return False;
        endif
      endif
    endfor
  endfor
  return False
endif
```

在该算法中，采用的是广度优先搜索算法对程序 P 进行基本块遍历[11]，同时选用后趋约束优先于前置约束的判定方式，这是由于后趋约束判定算法需要进行简单的相交基本块的搜寻，能较快速的进行对候选谓词的筛减。

4　实验与验证

4.1　原型系统设计

基于对广义不透明谓词的判定方法和检测过程，本文设计了基于约束求解的广义不透明谓词判定系统。该系统的主要构成包括：解密脱壳模块，静态分析模块，谓词判定模块，其中谓词判定模块是不透明谓词检测的主要功能部件，包括了前置条件约束提取组件，后趋逻辑约束提炼组件，约束求解谓词判定组件。组件的通联关系和运行流程如 4 图所示。

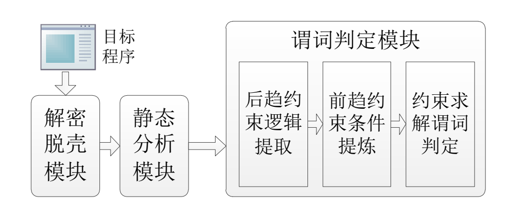


图 4　广义不透明谓词检测系统的构成

实验的开展首先是通过解密脱壳模块和静态分析模块将代码进行预处理，以还原出代码的执行内容。解密脱壳模块采用的 QuickUnpack 以及 OllyBone 工具，而静态分析模块是采用的 Angr[12]代码分析框架实现。Angr 是一款开源的二进制分析工具，能够提供两种控制流图的恢复方式，包括 CFGFast 和 CFGAccurate，其中 CFGFast 主要采用函数识别、递归分解、间接跳转解析的技术实现 CFG 恢复，CFGAccurate 则在 CFGFast 的基础上应用后向切片、符号执行等方法来提取更精准的 CFG。在本文的实验中，将选用 CFGAccurate 方法来获取代码的控制流图。

在广义不透明谓词检测中，谓词判定模块中前置条件约束提取和后趋逻辑约束提取采用的 K 和 L 基本块有界限制的长度分别为 16~24 以及 12~20；约束求解谓词判定使用的约束求解器为 Z3[13]。

基于上述方法实现的广义不透明谓词检测系统中，以 Angr 为基础的控制流图提取方法，能够对二进制的代码进行处理，在代码分析中具有普适性，并且静态的分析方法能够提高分析速率；而基于有界约束提取的方法可以将不透明性的判定限制在固定代码区间内，避免了在超大程序中过长的全代码分析耗时。所以本系统对程序的规模和语言并无特殊要求，具有较好的应用广度。

为了有效验证对广义不透明谓词的检测效率，本文将系统应用在对标准广义不透明谓词的识别和对实际混淆程序中广义不透明谓词的识别中。系统工作的基本软硬件环境为 4 核 3.1 GHz 处理器，8 GB 内存，Ubuntu 12.04 操作系统。

4.2　实验验证分析

从基于不透明谓词的判定方法中可以发现，主要的时间消耗将集中于三部分，一是在后趋逻辑约束的提取，二是对前置条件约束的提取，三是约束求解。其中约束求解部分的耗时最为不可控，对于约束求解，常会出现不可解的情况，在本文中，结合计算的精确和效率，将约束求解的时间限制为 5s，以确保不会出现死解的状态。

1）对自生成测试程序的检测

实验中首先针对自生成的包含广义不透明谓词的程序进行分析，广义不透明谓词将采用的是本文的示例程序和表一中不透明谓词样本，共计生成五个测试项。在对自生成程序的测试中，代码无须进行解密脱壳，直接进行静态分析，获取控制流图并开始谓词判定。对五个自生成程序的测试结果记录如下表所示。

表 2　对自生成程序的广义不透明谓词检测结果

谓词逻辑	程序规模	前置约束提取时间(ms)	后趋约束提取时间(ms)	平均约束求解时间(ms)	检出数	误报/漏报	测试总耗时平均值(ms)
$x+y==0$	23KB	39~48	5~8	19	1	0/0	211
x^2-$2x<2$	27KB	38~54	27~33	34	1	0/0	348
$x+y\le 3$	23KB	28~32	29~62	130	1	0/0	942
$x==k^2$	24KB	96~110	9~20	65	1	0/0	762
$x\%3==2$	31KB	49~52	22~29	3224	1	0/0	4003

由测试结果可见，在广义不透明谓词的检测中，本文设计的系统能够有效的检测出所有的插入的广义不透明谓词，其中，五次测试中均无误报和漏报。在系统的分析过程中，随着采用的前置和后趋的约束提取基本块有界限制，约束提取时间略有波动，但震荡范围不大，在所有的测试当中，对约束的提取时间均在 110ms 以内，提取的时间消耗可接受。主要的测试时间差距是在约束求解部分，对比发现，第五项测试程序的约束求解时间消耗较大，进一步分析求解的方法，可以发现，这是由于根据谓词表达式和前置条件约束，会将当前的输入分为两部分，一部分驱使程序沿路径 $\mathcal{B}^T$ 执行，另一部分驱使程序沿路径 $\mathcal{B}^F$ 执行，在第五项的输入分类中，两个驱使执行的输入集都比较大，进行约束表达式求解时，求解器遍历的集合空间较大，这就导致了测试消耗明显高于其他测试项。总体看来，整个测试的耗时属于可接受的范围。

2）对实例恶意代码程序的检测

为了更好的验证本文提出的基于代码逻辑的广义不透明谓词检测方法，将选用多型恶意代码程序样本进行实例分析，以检验方法的有效性。作为一种常用的代码混淆方法，不透

明谓词已经被广泛应用在各类恶意代码中。在此，文章从 VX Heavens 提供的恶意代码库中，选取了 10 款较为典型的恶意代码样本，这些样本涵盖了病毒、木马、后门、蠕虫四种类型的多型变种。同时为了验证检测出来的不透明谓词的正确性，本文选用的样本将具有对应的源代码，或者有详细的样本分析报告，以确保能进行正确性比对。测试的结果显示在表 3 中。

表 3 对典型恶意代码样本的广义不透明谓词检测结果

样本类别	类型	规模(KB)	狭义OP	广义OP	误报	漏报	平均单次约束提取时间(ms)	平均单次约束求解时间(ms)	测试耗时(s) 广义	测试耗时(s) 广义和狭义
Tefuss	Virus	29	7	0	1/0	0/0	103	624	4.1	38.9
Zmist	Virus	30	13	1	0/0	1/0	76	431	12.9	70.6
Ranx	Backdoor	53	8	3	0/1	0/0	210	1057	15.6	56.6
RBot	Backdoor	62	12	1	0/0	0/0	56	227	13.2	141.2
Rub	Trojan	20	5	1	0/0	0/0	70	199	20.2	312.3
SalityStub	Trojan	82	21	2	0/0	1/0	142	326	49.4	410.0
Reper	Trojan	38	5	0	0/0	0/0	91	514	16.7	221.8
Netsky	Worm	26	11	0	2/0	0/0	127	1314	6.1	97.9
Klez	Worm	43	12	1	0/0	0/0	69	480	9.0	79.1
Mydoom	Worm	41	7	2	0/0	0/0	90	891	18.3	177.4

狭义不透明谓词的检测与广义不透明谓词的检测方法基本一致，也是采用约束求解的方法判定谓词表达式的恒值性，在测试中，将同样进行针对性的检测。表 3 很好的展示了对不透明谓词的检测结果和检测耗时，其中第四列和第五列标明了在分析过程中，检测出的狭义不透明谓词和广义不透明谓词的数量；结合样本对应的源代码或分析报告，罗列出检测中出现的误报和漏报不透明谓词数目；并在第八列和第九列中，展示了平均单次约束提取的时间和平均单次约束求解的时间；整个测试项目的耗时如表中最后两列所示，其中包括了对广义不透明谓词的测试耗时，以及对广义即狭义不透明谓词的测试耗时。从测试的结果可以发现，在恶意代码中，均多次插入了狭义不透明谓词，而广义不透明谓词也在多数恶意代码中得到了运用。进一步分析，在不透明谓词的检测中，存在少量的误报和漏报，这主要是由于在对代码的静态分析中，受到了平展控制流，自修改代码[14]等混淆手段的干扰，使得对不透明谓词的分析中提取的信息缺失，从而导致测试中的遗漏和错误。由测试时间可得，当进行广义不透明谓词和狭义不透明谓词的同时检测时，其耗时明显多于广义不透明谓词的单独测试时间，这是由于广义不透明分析的测试，采用了先找寻相交基本块，再进行谓词约束提取和求解的层次化分析方法，能较好的对测试项进行初筛，而狭义不透明谓词的测试，需要在每个跳转位置进行直接的约束求解，测试项相对较多。

在对实例恶意代码的测试中，本文提出的广义不透明谓词检测方法能够快速准确的测试出当前代码中引入的广义不透明谓词，并且采用了层次化的谓词约束求解方法，能够很好的实现低耗时的测试。

5 结束语

不透明谓词是控制流图混淆的关键应用技术，通过引入多余的分支路径实现对程序分析的干扰，并作为一种重要的防查杀手段被广泛应用于恶意代码的生存性强化中。广义不透明谓词不同于狭义不透明谓词，其恒定不变的属性由值恒定扩展为逻辑恒定，具有更强的抵御逆向分析的能力。本文分析了广义不透明谓词的特点，提取了广义不透明谓词中前置条件约束和后趋逻辑约束，并结合谓词表达式，综合检测广义不透明谓词。其中为了提升检测的效能，采用了基于相交基本块的层次化初筛方法，减少检测实例，对于通过初筛的谓词表达式，采取约束求解的方式来验证谓词逻辑的恒定性。实验验证可知，本文提出的方法能快速准确的检测出广义不透明谓词，并在实际恶意代码检测中具有较强的实用性。

本文实现的不透明谓词检测系统在应对未知壳，以及平展控制流和自修改代码等动态代码混淆技术时，在控制流图提取上存在一定困难。故文章下一步将采取动态分析[15]的手段，规避加密加壳对程序分析的影响，进一步提升本文检测系统在应对动态混淆代码时的检测能力。

参考文献：

[1] Cipresso T, Stamp M. Software Reverse Engineering [M]// Handbook of Information and Communication Security. Berlin: Springer, 2010.

[2] 韩翔宇, 李强, 黄海军, 等. 基于不透明谓词的软件抗动态逆向分析研究 [J]. 计算机应用研究, 2017, 34 (8): 2422-2428.

[3] Balachandran V, Sufatrio, Tan D J J, *et al.* Control flow obfuscation for Android applications [J]. Computers & Security, 2016, 61: 72-93.

[4] Collberg C, Thomborson C, Low D. A taxonomy of obfuscating

transformations [R]. Department of Computer Science the University of Auckland New Zealand, 1997.

[5] 吴伟民, 林水明, 余国鹏, 等. 基于哈希不透明谓词的 JavaScript 软件水印算法 [J]. 计算机应用与软件, 2016, 33 (4): 306-309.

[6] Preda M D, Madou M, Bosschere K D, *et al.* Opaque predicates detection by abstract interpretation [C]// Lecture Notes in Computer Science, vol 4019. 2006: 81-95.

[7] Saleh M, Ratazzi E P, Xu S. Instructions-based detection of sophisticated obfuscation and packing [C]// Proc of Military Communications Conference. 2015: 1-6.

[8] Bardin S, David R, Marion J Y. Backward-bounded DSE: targeting infeasibility questions on obfuscated codes [C]// Security and Privacy. 2017: 633-651.

[9] Ming J, Xu D, Wang L, *et al.* LOOP: logic-oriented opaque predicate detection in obfuscated binary code [C]// Proc of ACM SIGSAC Conference on Computer and Communications Security. New York: ACM Press, 2015: 757-768.

[10] Franzén A, Cimatti A, Nadel A, *et al.* Applying SMT in symbolic execution of microcode [C]// Formal Methods in Computer-Aided Design. 2011: 121-128.

[11] 秦晓军, 周林, 陈左宁, 等. 基于懒符号执行的软件脆弱性路径求解算法 [J]. 计算机学报, 2015, 38 (11): 2290-2300.

[12] Yan S, Wang R, Salls C, *et al.* SOK: (state of) the art of war: offensive techniques in binary analysis [C]// Security and Privacy. 2016: 138-157.

[13] Moura L D, Bjørner N. Z3: an efficient SMT solver [M]// Tools and Algorithms for the Construction and Analysis of Systems. Berlin: Springer, 2008: 337-340.

[14] 何炎祥, 陈勇, 吴伟, 等. 基于程序流敏感的自修改代码混淆方法 [J]. 计算机工程与科学, 2012, 34 (1): 79-85.

[15] 马洪亮, 王伟, 韩臻. 混淆恶意 JavaScript 代码的检测与反混淆方法研究 [J]. 计算机学报, 2017, 40 (7): 1699-1713.